JSP - EXPRESSION LANGUAGE (EL)

JSP Expression Language (EL) makes it possible to easily access application data stored in JavaBeans components. JSP EL allows you to create expressions both **(a)** arithmetic and **(b)** logical. Within a JSP EL expression, you can use integers, floating point numbers, strings, the built-in constants true and false for boolean values, and null.

Simple Syntax:

Typically, when you specify an attribute value in a JSP tag, you simply use a string. For example:

```
<jsp:setProperty name="box" property="perimeter" value="100"/>
```

JSP EL allows you to specify an expression for any of these attribute values. A simple syntax for JSP EL is as follows:

```
${expr}
```

Here **expr** specifies the expression itself. The most common operators in JSP EL are **.** and **[]**. These two operators allow you to access various attributes of Java Beans and built-in JSP objects.

For example above syntax <jsp:setProperty> tag can be written with an expression like:

```
<jsp:setProperty name="box" property="perimeter"
                 value="${2*box.width+2*box.height}"/>
```

When the JSP compiler sees the ${} form in an attribute, it generates code to evaluate the expression and substitues the value of expresson.

You can also use JSP EL expressions within template text for a tag. For example, the <jsp:text> tag simply inserts its content within the body of a JSP. The following <jsp:text> declaration inserts <h1>Hello JSP!</h1> into the JSP output:

```
<jsp:text>
<h1>Hello JSP!</h1>
</jsp:text>
```

You can include a JSP EL expression in the body of a <jsp:text> tag (or any other tag) with the same ${} syntax you use for attributes. For example:

```
<jsp:text>
Box Perimeter is: ${2*box.width + 2*box.height}
</jsp:text>
```

EL expressions can use parentheses to group subexpressions. For example, ${(1 + 2) * 3} equals 9, but ${1 + (2 * 3)} equals 7.

To deactivate the evaluation of EL expressions, we specify the isELIgnored attribute of the page directive as below:

```
<%@ page isELIgnored ="true|false" %>
```

The valid values of this attribute are true and false. If it is true, EL expressions are ignored when they appear in static

text or tag attributes. If it is false, EL expressions are evaluated by the container.

Basic Operators in EL:

JSP Expression Language (EL) supports most of the arithmatic and logical operators supported by Java. Below is the list of most frequently used operators:

Operator	Description
.	Access a bean property or Map entry
[]	Access an array or List element
( )	Group a subexpression to change the evaluation order
+	Addition
-	Subtraction or negation of a value
*	Multiplication
/ or div	Division
% or mod	Modulo (remainder)
== or eq	Test for equality
!= or ne	Test for inequality
< or lt	Test for less than
> or gt	Test for greater than
<= or le	Test for less than or equal
>= or gt	Test for greater than or equal
&& or and	Test for logical AND
|| or or	Test for logical OR
! or not	Unary Boolean complement
empty	Test for empty variable values

Functions in JSP EL :

JSP EL allows you to use functions in expressions as well. These functions must be defined in custom tag libraries. A function usage has the following syntax:

```
${ns:func(param1, param2, ...)}
```

Where ns is the namespace of the function, func is the name of the function and param1 is the first parameter value. For example, the function fn:length, which is part of the JSTL library can be used as follows to get the the length of a string.

```
${fn:length("Get my length")}
```

To use a function from any tag library (standard or custom), you must install that library on your server and must include the library in your JSP using <taglib> directive as explained in JSTL chapter.

JSP EL Implicit Objects:

The JSP expression language supports the following implicit objects:

Implicit object	Description
pageScope	Scoped variables from page scope
requestScope	Scoped variables from request scope
sessionScope	Scoped variables from session scope
applicationScope	Scoped variables from application scope
param	Request parameters as strings
paramValues	Request parameters as collections of strings
header	HTTP request headers as strings
headerValues	HTTP request headers as collections of strings
initParam	Context-initialization parameters
cookie	Cookie values
pageContext	The JSP PageContext object for the current page

You can use these objects in an expression as if they were variables. Here are few examples which would clear the concept:

The pageContext Object:

The pageContext object gives you access to the pageContext JSP object. Through the pageContext object, you can access the request object. For example, to access the incoming query string for a request, you can use the expression:

```
${pageContext.request.queryString}
```

The Scope Objects:

The pageScope, requestScope, sessionScope, and applicationScope variables provide access to variables stored at each scope level.

For example, If you need to explicitly access the box variable in the application scope, you can access it through the applicationScope variable as applicationScope.box.

The param and paramValues Objects:

The param and paramValues objects give you access to the parameter values normally available through the request.getParameter and request.getParameterValues methods.

For example, to access a parameter named order, use the expression ${param.order} or ${param["order"]}.

Following is the example to access a request parameter named username:

```
<%@ page import="java.io.*,java.util.*" %>
<%
    String title = "Accessing Request Param";
%>
<html>
<head>
<title><% out.print(title); %></title>
</head>
<body>
<center>
<h1><% out.print(title); %></h1>
</center>
<div align="center">
<p>${param["username"]}</p>
</div>
</body>
</html>
```

The param object returns single string values, whereas the paramValues object returns string arrays.

header and headerValues Objects:

The header and headerValues objects give you access to the header values normally available through the request.getHeader and request.getHeaders methods.

For example, to access a header named user-agent, use the expression ${header.user-agent} or ${header["user-agent"]}.

Following is the example to access a header parameter named user-agent:

```
<%@ page import="java.io.*,java.util.*" %>
<%
    String title = "User Agent Example";
%>
<html>
<head>
<title><% out.print(title); %></title>
</head>
<body>
<center>
<h1><% out.print(title); %></h1>
</center>
<div align="center">
<p>${header["user-agent"]}</p>
</div>
</body>
</html>
```

This would display something as follows:

USER AGENT EXAMPLE

Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; WOW64; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; HPNTDF; .NET4.0C; InfoPath.2)

The header object returns single string values, whereas the headerValues object returns string arrays.